

Operating Systems and Kernel Design

Lab Assignment -4 Report

Pavan Kumar Revanuru

01578776



Due Date: 11/27/2016

Objective:

The objective of the assignment is to implement a simple client-server model to exchange commands over a network, to mimic some part of the functionality of a distributed scheduler and RPC.

Functions and methods used:

getpid():

getpid() gives the Process ID number of the current process. The process ID is limited only to that process and if perfectly executed, the process get killed when exiting.

fork():

fork() creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process. The child process and the parent process run in separate memory spaces. At the time of fork() both memory spaces have the same content. Memory writes, file mappings performed by one of the processes do not affect the other.

wait():

wait() is used for the state change in a child of the calling process. This gives a chance to obtain information about the child if it's state has changed. If there is a state change, then it can be confirmed that the child process has been killed and the program can continue its rest of the executing. This is done to avoid creating Zombies. (The process which won't die).

execv():

execv() function provides an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the filename associated with the file being executed.

socket(AF_INET,SOCK_STREAM,0):

This function is used to create sockets. This is used in various parts of the code to create many sockets.

Memset():

The memset() function fills the first n bytes of the memory area pointed to with the constant byte.

Bind():

When a socket is created with socket(2), it exists in a name space (address family) but has no address assigned to it. bind() assigns the address specified by addr to the socket referred to by the file descriptor sockfd. addrlen specifies the size, in bytes, of the address structure pointed to by addr. Traditionally, this operation is called "assigning a name to a socket".

Recv():

The recv() calls are used to receive messages from a socket. It may be used to receive data on both connectionless and connection-oriented sockets.

Send():

The system call send() is used to transmit a message to another socket.

Strncpy():

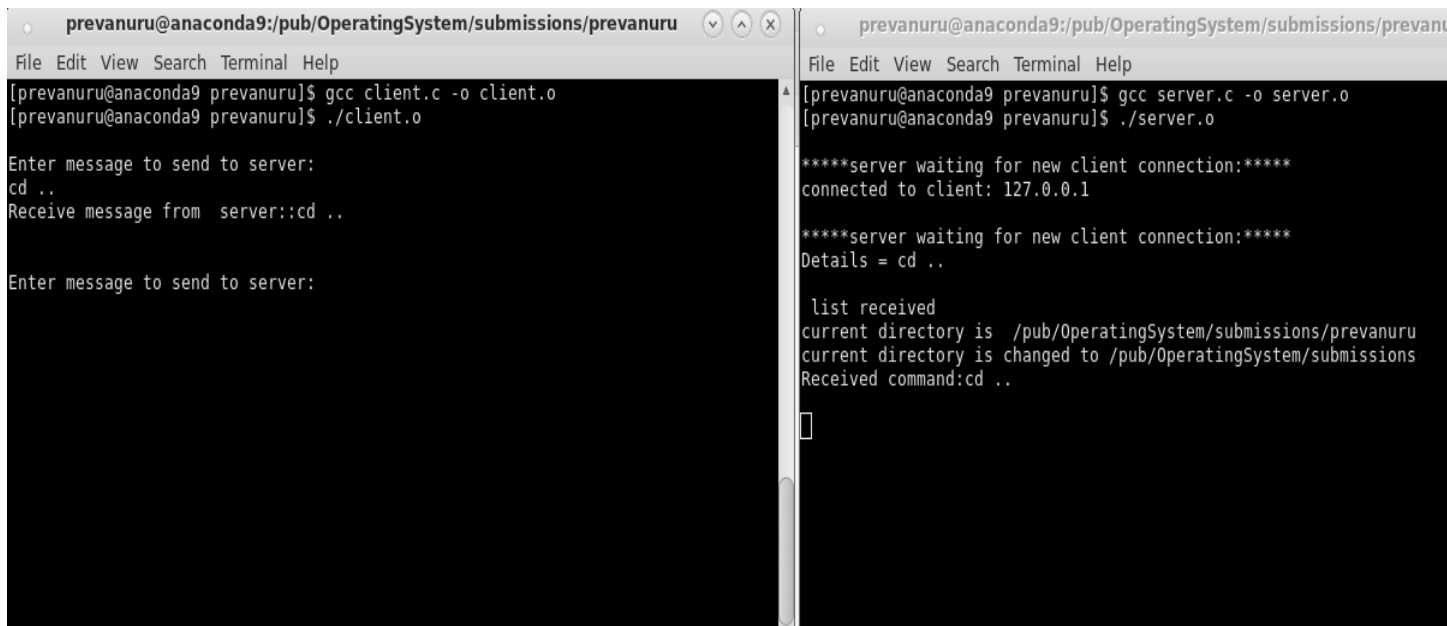
The strncpy() function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy.

Background of the assignment:

The assignment focuses on creating the client and server programs. It includes creating many processes, for each and every client connected to the server. It also focuses on commands executed with the help of execve .

Results:

Change Directory(cd):



```
prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru
File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]$ gcc client.c -o client.o
[prevanuru@anaconda9 prevanuru]$ ./client.o

Enter message to send to server:
cd ..
Receive message from server::cd ..

Enter message to send to server:

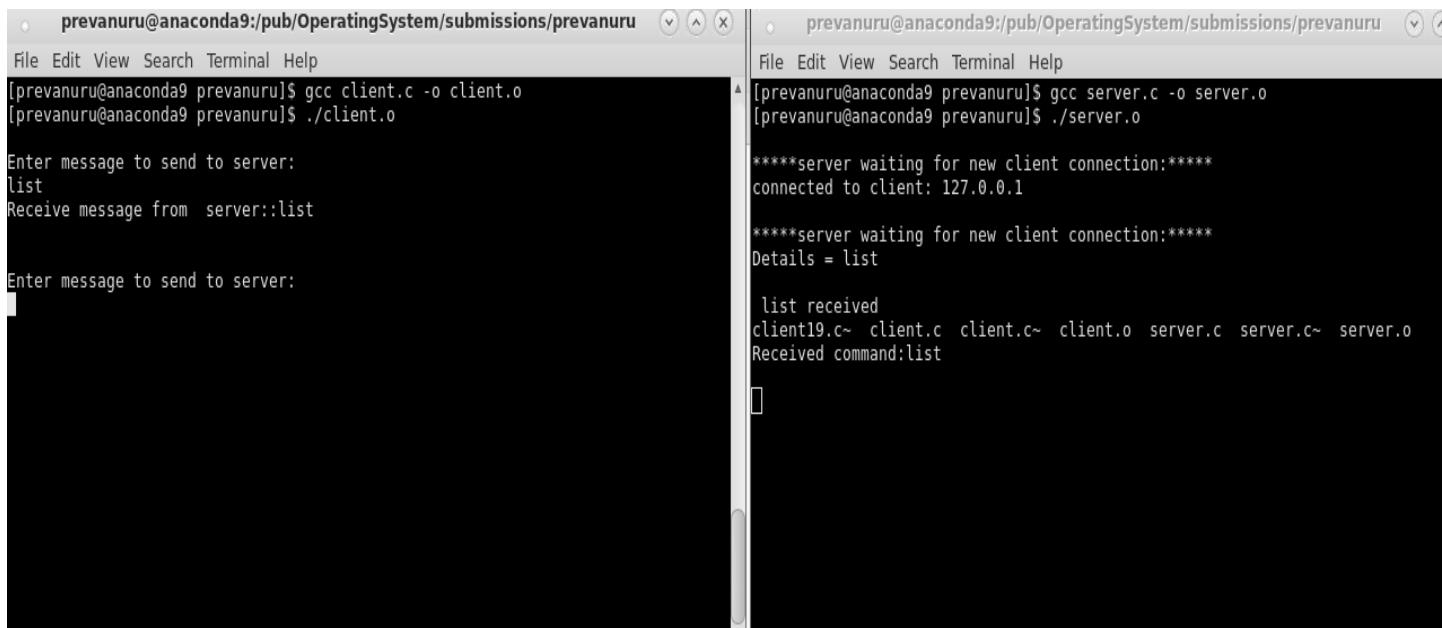
prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru
File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]$ gcc server.c -o server.o
[prevanuru@anaconda9 prevanuru]$ ./server.o

****server waiting for new client connection:****
connected to client: 127.0.0.1

****server waiting for new client connection:****
Details = cd ..

list received
current directory is /pub/OperatingSystem/submissions/prevanuru
current directory is changed to /pub/OperatingSystem/submissions
Received command:cd ..
```

List(ls):



The image shows two terminal windows side-by-side. The left window is the client terminal, and the right window is the server terminal. Both are running on a system with the username 'prevanuru' and the directory '/pub/OperatingSystem/submissions/prevanuru'.

Client Terminal (Left):

```
prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru
File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]$ gcc client.c -o client.o
[prevanuru@anaconda9 prevanuru]$ ./client.o

Enter message to send to server:
list
Receive message from server::list

Enter message to send to server:

```

Server Terminal (Right):

```
prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru
File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]$ gcc server.c -o server.o
[prevanuru@anaconda9 prevanuru]$ ./server.o

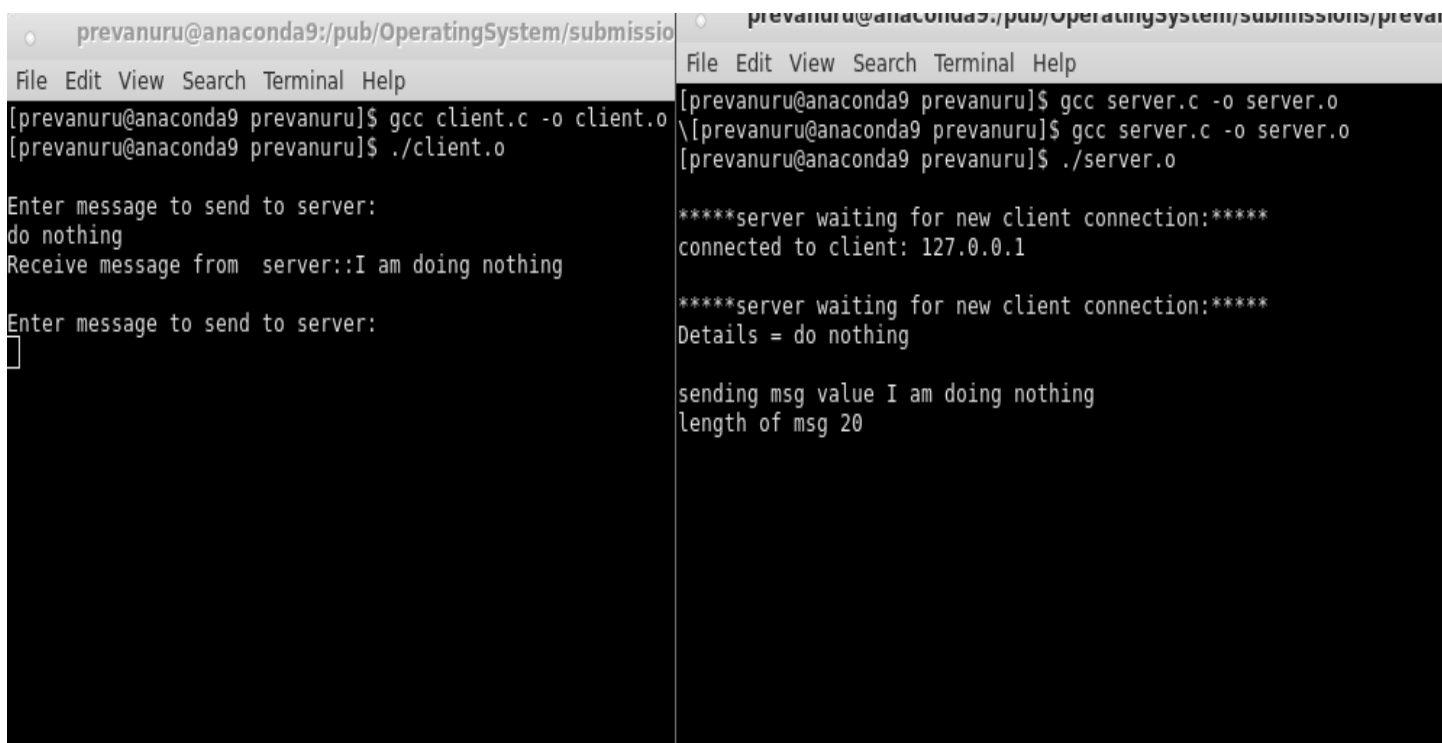
****server waiting for new client connection:****
connected to client: 127.0.0.1

****server waiting for new client connection:****
Details = list

list received
client19.c~ client.c client.c~ client.o server.c server.c~ server.o
Received command:list

```

Do nothing:



The image shows two terminal windows side-by-side. The left window is the client terminal, and the right window is the server terminal. Both are running on a system with the username 'prevanuru' and the directory '/pub/OperatingSystem/submissions/prevanuru'.

Client Terminal (Left):

```
prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru
File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]$ gcc client.c -o client.o
[prevanuru@anaconda9 prevanuru]$ ./client.o

Enter message to send to server:
do nothing
Receive message from server::I am doing nothing

Enter message to send to server:

```

Server Terminal (Right):

```
prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru
File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]$ gcc server.c -o server.o
[prevanuru@anaconda9 prevanuru]$ gcc server.c -o server.o
[prevanuru@anaconda9 prevanuru]$ ./server.o

****server waiting for new client connection:****
connected to client: 127.0.0.1

****server waiting for new client connection:****
Details = do nothing

sending msg value I am doing nothing
length of msg 20

```

Quit():

prevanuru@anaconda9:/pub/OperatingSystem/submission	prevanuru@anaconda9:/pub/OperatingSystem/submissions/pr
File Edit View Search Terminal Help	File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]\$ gcc client.c -o client.o	[prevanuru@anaconda9 prevanuru]\$ gcc server.c -o server.o
[prevanuru@anaconda9 prevanuru]\$./client.o	[prevanuru@anaconda9 prevanuru]\$ gcc server.c -o server.o
	[prevanuru@anaconda9 prevanuru]\$./server.o
Enter message to send to server:	****server waiting for new client connection:****
do nothing	connected to client: 127.0.0.1
Receive message from server::I am doing nothing	
	****server waiting for new client connection:****
Enter message to send to server:	Details = do nothing
quit	
closing socket	sending msg value I am doing nothing
[prevanuru@anaconda9 prevanuru]\$	length of msg 20
	Details = quit

Two clients connected to one server:

prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru	prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru
File Edit View Search Terminal Help	File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]\$ gcc client.c -o client.o	****server waiting for new client connection:****
[prevanuru@anaconda9 prevanuru]\$./client.o	Details = do nothing
	sending msg value I am doing nothing
Enter message to send to server:	length of msg 20
do nothing	Details = quit
Receive message from server::I am doing nothing	
	connected to client: 127.0.0.1
Enter message to send to server:	****server waiting for new client connection:****
quit	connected to client: 127.0.0.1
closing socket	****server waiting for new client connection:****
[prevanuru@anaconda9 prevanuru]\$./client.o	Details = hi
	sending msg value What?
Enter message to send to server:	length of msg 7
hi	Details = list
Receive message from server::What?	
	list received
Enter message to send to server:	client19.c~ client.c client.c~ client.o server.c server.c~
	Received command:list

prevanuru@anaconda9:/pub/OperatingSystem/submissions/prevanuru
File Edit View Search Terminal Help
[prevanuru@anaconda9 prevanuru]\$./client.o
Enter message to send to server:
list
Receive message from server::list
Enter message to send to server:

Observations:

The client and server architecture is the concurrent model that is implemented. The server takes many commands such as ls, cd, quit and do nothing. The ls command is executed when 'list' is sent on the client side. 'cd' is executed when the entire command "cd .." or "cd /" is given. Do nothing gives back a normal response from the server. The quit command closes the socket for one particular client.

Conclusion:

The programs executed are:

1. Created a concurrent server which can accept many client at once on First come first services.
2. The client gives commands to the server and server services the commands.
3. If the client sends an invalid command such as "hi", the server responds with "What?".

Source Code:

Server.c

```
#include<stdio.h>

#include<sys/types.h>//socket
#include<sys/socket.h>//socket
#include<string.h>//memset
#include<stdlib.h>//sizeof
#include<netinet/in.h>//INADDR_ANY
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define PORT 47004
#define MAXSZ 100
int main()
{
    int sockfd;//to create socket
```

```
int newsockfd;//to accept connection

//int pid;

struct sockaddr_in serverAddress;//server receive on this address
struct sockaddr_in clientAddress;//server sends to client on this address


int n;
char msg[MAXSZ];


int clientAddressLength;
int pid;


//create socket
sockfd=socket(AF_INET,SOCK_STREAM,0);
//initialize the socket addresses
memset(&serverAddress,0,sizeof(serverAddress));
serverAddress.sin_family=AF_INET;
serverAddress.sin_addr.s_addr=htonl(INADDR_ANY);
serverAddress.sin_port=htons(PORT);


//bind the socket with the server address and port
bind(sockfd,(struct sockaddr *)&serverAddress, sizeof(serverAddress));


//listen for connection from client
listen(sockfd,5);


while(1)
{
    //parent process waiting to accept a new connection
    printf("\nserver waiting for new client connection:\n");
```

```
clientAddressLength=sizeof(clientAddress);
newsockfd=accept(sockfd,(struct sockaddr*)&clientAddress,&clientAddressLength);
printf("connected to client: %s\n",inet_ntoa(clientAddress.sin_addr));
```

```
//child process is created for serving each new clients
```

```
pid=fork();
```

```
if(pid==0)//child process rec and send
```

```
{
```

```
    //receive from client
```

```
    while(1)
```

```
    {
```

```
        n=recv(newsockfd,msg,MAXSZ,0);
```

```
        if(n==0)
```

```
        {
```

```
            close(newsockfd);
```

```
            break;
```

```
        }
```

```
        printf("Details = %s\n", msg);
```

```
        if(msg[0]=='l'&& msg[1]=='i'&& msg[2] == 's' && msg[3] == 't')
```

```
        {
```

```
            printf(" list received\n");
```

```
            int a;
```

```
            pid = fork();
```

```
            if(pid == 0)
```

```
            {
```

```
                char *args[] = {"ls", NULL};
```

```
                a = execv("/bin/ls", args);
```

```
            exit(1);
```



```

    }
    else
    {
        wait(NULL);
    }
}

```

```

else if(msg[0]=='c' && msg[1]=='d')
{
    printf(" list received\n");
    int a,i=0;
    char point[MAXSZ];

    printf("current directory is %s \n", getcwd(point,MAXSZ));

    char directory[MAXSZ] = "/";
    char new_dire[MAXSZ];

    strncpy(directory,msg+3,strlen(msg));

    //printf("directory is %s\n", directory);
    //printf("size of msg is %d\n",strlen(directory));
    strncpy(new_dire,directory,strlen(directory)-1);
    //printf("New directory is %s\n", new_dire);
    //printf("New size of msg is %d\n",strlen(new_dire));
    a = chdir(new_dire);
    //printf("a value is %d\n", a );
    //point = getcwd(point,MAXSZ);
    printf("current directory is changed to %s \n", getcwd(point,MAXSZ));
}

```

```

}

else if(msg[0] == 'd' && msg[1] == 'o' && msg[2] == ' ' && msg[3] == 'n')
{
    char dest[MAXSZ] = "I am doing nothing ";
    //strcpy(dest,"I am doing nothing");
    printf("sending msg value %s\n", dest);
    printf("length of msg %d\n", strlen(dest));
    int i = strlen(dest);
    dest[i]=0;
    send(newsockfd,dest,i,0);
    continue;

}

else if(msg[0] == 'q' && msg[1] == 'u')
{
    close(newsockfd);
    exit(1);
}

else
{
    char dest[MAXSZ] = "What? ";
    //strcpy(dest,"I am doing nothing");
    printf("sending msg value %s\n", dest);
    printf("length of msg %d\n", strlen(dest));
    int i = strlen(dest);

```

```

        dest[i]=0;

        send(newsockfd,dest,i,0);

        continue;

    }

    msg[n]=0;

    send(newsockfd,msg,n,0);

    printf("Received command:%s\n",msg);


    }//close interior while
    exit(0);
}

else
{
    close(newsockfd);//sock is closed BY PARENT
}

}

}

return 0;

}

```

Client.c:

```

#include<stdio.h>

#include<sys/types.h>//socket

#include<sys/socket.h>//socket

```

```

#include<string.h>//memset
#include<stdlib.h>//sizeof
#include<netinet/in.h>//INADDR_ANY
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 47004
#define SERVER_IP "127.0.0.1"
#define MAXSZ 100

int main()
{
    int sockfd;//to create socket

    struct sockaddr_in serverAddress;//client will connect on this

    int n;
    char msg1[MAXSZ];
    char msg2[MAXSZ];

    //create socket
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    //initialize the socket addresses
    memset(&serverAddress,0,sizeof(serverAddress));
    serverAddress.sin_family=AF_INET;
    serverAddress.sin_addr.s_addr=inet_addr(SERVER_IP);
    serverAddress.sin_port=htons(PORT);

    //client connect to server on port
    connect(sockfd,(struct sockaddr *)&serverAddress,sizeof(serverAddress));

```

```

//send to sever and receive from server
while(1)
{
    printf("\nEnter message to send to server:\n");
    fgets(msg1,MAXSZ,stdin);
    if(msg1[0]=='#')
        break;

    n=strlen(msg1)+1;
    send(sockfd,msg1,n,0);
    if(msg1[0]=='q' && msg1[1]=='u')
    {
        printf("closing socket\n");
        close(sockfd);
        break;
    }
    n=recv(sockfd,msg2,MAXSZ,0);

    printf("Receive message from server::%s\n",msg2);
    //printf("length of received msg is %d\n",strlen(msg2));
}

return 0;
}

```